Computer Networks 82 (2015) 81–95

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Multihybrid job scheduling for fault-tolerant distributed computing in policy-constrained resource networks

Yong-Hyuk Moon [a,*], Chan-Hyun Youn [b]

[a] Electronics and Telecommunications Research Institute, South Korea
[b] Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, South Korea

ARTICLE INFO

Article history:
Received 17 June 2014
Received in revised form 1 December 2014
Accepted 4 February 2015
Available online 10 March 2015

Keywords:
Job scheduling
Fault tolerance
Policy heterogeneity
Multiobjective optimization
Distributed computing
Genetic algorithm

ABSTRACT

Unpredictable fluctuations in resource availability often lead to rescheduling decisions that sacrifice a success rate of job completion in batch job scheduling. To overcome this limitation, we consider the problem of assigning a set of sequential batch jobs with demands to a set of resources with constraints such as heterogeneous rescheduling policies and capabilities. The ultimate goal is to find an optimal allocation such that performance benefits in terms of makespan and utilization are maximized according to the principle of Pareto optimality, while maintaining the job failure rate close to an acceptably low bound. To this end, we formulate a multihybrid policy decision problem (MPDP) on the primary-backup fault tolerance model and theoretically show its NP-completeness. The main contribution is to prove that our multihybrid job scheduling (MJS) scheme confidently guarantees the fault-tolerant performance by adaptively combining jobs and resources with different rescheduling policies in MPDP. Furthermore, we demonstrate that the proposed MJS scheme outperforms the five rescheduling heuristics in solution quality, searching adaptability and time efficiency by conducting a set of extensive simulations under various scheduling conditions.

1. Introduction

One of the most challenging requirements of job scheduling systems currently being developed is ensuring that they elastically allocate computing resources to jobs despite the unpredictable occurrence of resource failures. With increased complexity, dynamic features, and various uncertainties, resource failure [1–3] is the rule rather than the exception in distributed computing systems (DCS) such as grid, peer-to-peer, and cloud computing. It has been reported that over 75% and 70% of the resources have failure rates of about 20% and 40% in workload archives such as DEUG, and UCB and SDSC [3], respectively. From these application-level traces, we found that most resources have relatively high failure probabilities. It is also recognized that failures can significantly affect scheduling performance and that the large number of job failures is still caused by resource fluctuations and unavailability, as discussed in [2]. Specifically, Ref. [4] has exhibited the statistical overviews on job execution in ten different DCS in which the proportion of failed jobs varied from 1.23% to 83.94% and failed jobs consume lots of computational power of resources, ranging from 0.41% to 73.88%. Although most resources are managed by autonomous domains (ADs), each AD tends to employ proprietary administrative rules (i.e., rescheduling policies) for job recovery. This fact further complicates the large scale resource pooling for reliable job execution in a cooperative manner. As a result, in addition to extending known

* Corresponding author.
E-mail addresses: yhmoon@etri.re.kr (Y.-H. Moon), chyoun@kaist.ac.kr (C.-H. Youn).

http://dx.doi.org/10.1016/j.comnet.2015.02.030
1389-1286/

rescheduling policies, it requires a new scheduling algorithm that can efficiently guarantee fault tolerance.

There are some scheduling approaches recently studied in order to alleviate the undesirable effects of resource failures with different policies. In [5], a task re-execution policy employed in Hadoop has been currently studied in order to analyze its impact on job completion reliability and time from a theoretical viewpoint. However, it is not clear which configuration of this policy is sufficient to prevent a running job from being interrupted by resource failures. To overcome this limitation, Zhang et al. [6] have integrated backfilling and migration with well-established gang scheduling strategy. The fully integrated allocation method consistently outperforms the others. Nevertheless, the authors have not explicitly addressed what specific conditions can decide the most suitable combination of different strategies. For similar reason, a prior study [7] proposes a concept of dynamic policy switching at run time for interactive jobs, which require immediate execution. Its weakness is that the proposed scheme is irrelevant for batch jobs that are submitted to a job-ready queue and await their turn to run. On the contrary to that work, in [8] the authors propose how to decide a near-optimal size of hybrid cluster in clouds for accelerating batch analytics. From this study, we have found that it is possible to tolerate a high degree of instability in clouds if system-level metrics are only given; however, the degree of performance gain is not strongly correlated with the resource usage cost. To tackle this problem, Farahabady el al. [9] proposes a new scheduling algorithm which generates solutions with the Pareto optimality between cost and performance. The key question of how to dynamically cope with performance variability of underlying resources due to failures remains unanswered. Furthermore, the performance impacts of different combinations of job selection policies and job dispatching mechanisms have been presented for the Bag-of-Tasks (BoT) allocation [10]. Although this study has shown monotonic improvement in some cases, they have only emphasized the static combinations of different job allocation strategies and have paid less attention to the fact that resources are controlled by heterogeneous rules. In [11], the authors have analyzed the correlation between job sizes and scheduling algorithms. However, they have not considered the potential gains of using different scheduling approaches simultaneously. Besides, the failure condition has not been considered a major issue.

To the best of our knowledge, there have not been direct contributions to tackle the policy heterogeneity problem in the context of job scheduling. Thus, a new policy-integrated scheduling scheme needs to answer the following research questions, which have remained unsolved: (1) How much performance improvement is expected, as compared to static rescheduling policies? (2) How much fault tolerance can it provide without sacrificing the main objectives considering a large fraction of resources in availability? (3) Assuming that different policies are supported in various ADs, how can it adapt to this situation regarding scheduling quality and complexity? In this paper, we propose a multihybrid job scheduling (MJS) scheme in order to deliver robust resource allocation for computational batch jobs under heterogeneous policies. The proposed MJS scheme adopts a messy genetic algorithm (mGA) [12], which has been applied to solve combinatorial optimization problems. The main contributions of this study are as follows:

- We formulate the aforementioned problem as a multihybrid policy decision problem (MPDP) on the primary-backup fault tolerance model and discuss how an optimal solution can be quantified in terms of scheduling quality.
- To solve MPDP effectively, we propose a new MJS scheme that finds an optimal schedule even in a large search space by using stochastic search operations of mGA within relatively low and acceptable complexity.
- The mGA-based MJS scheme demonstrates high fault tolerance without sacrificing the other objectives, such as makespan and load balance, unlike the static approaches and deterministic algorithms (e.g., *minmin* [13]), even in the policy-constrained DCS.

The remainder of this paper is organized as follows. Section 2 presents a fault-tolerant job scheduling model with the four rescheduling policies. We formulate MPDP in Section 3 and then propose an mGA-based MJS scheme, focusing on solution representation and realization in Section 4. Section 5 provides analytical views on the terms of approximation of computational complexity and possibility of convergence of the proposed scheme. Simulation results and discussions are in Section 6. Finally, we conclude the paper in Section 7.

2. Fault-tolerant job scheduling model

A job scheduling system in DCS consists of three tiers: users submitting batch jobs $\mathcal{J} = \{j_i | i = 1, 2, 3, \ldots c, N\}$, a job scheduler, and computing resources $\mathcal{R} = \{r_x | x = 1, 2, 3, \ldots, M\}$. Each resource x, r_x, has its relative computational speed c_x, which is assumed to be an arbitrary and non-decreasing (i.e., concave) function over time. A job scheduler distributes stochastic workloads $\nu(\Delta t)$ arriving in a given period of time, Δt across allocated resources. An arriving j_i can be distinguished by its arrival time $\tilde{a}_i$, job length l_i, and deadline $\tilde{d}_i$. $\tilde{a}_i$ should be bounded to $[0, \Delta t]$ and l_i indicates the time spent by r_x with $c_x = 1$ (i.e., reference computing element) to execute j_i. Therefore, the effective length of j_i can be calculated as $\tilde{l}_{i,x} = l_i / c_x$, if there is no failure. We suppose that at time zero, all resources are operable and jobs can be allocated to them. Resources can execute, at most, a single job at a time and can also fail at any random time. Table 1 summarizes a number of commonly used symbols and we will discuss some of them further after that.

2.1. Application workload

Realistic workloads comprise mostly single-process jobs, and 85–95% of them are batch jobs as measured in the several grid workloads [14]. Likewise, we adopt the

Table 1
Definition of symbols used in the mathematical model.

Symbols	Definition
$\mathcal{J},\ \mathcal{R},\ \mathbf{\Pi}$	A set of jobs, available resources, and rescheduling policies
$j_i \subseteq \mathcal{J},\ r_x \subseteq \mathcal{R},\ \Pi_x \subset \mathbf{\Pi}$	ith job, xth resource, and rescheduling policies supported by r_x
$l_i,\ c_x,\ \tilde{d}_i,\ \tilde{a}_i,\ \tilde{l}_{i,x}$	A length of j_i, a computational speed of r_x, a deadline of j_i, an arrival time of j_i, and a effective length of j_i allocated to r_x
$\tilde{r}_i,\ \tilde{c}_i,\ \tilde{s}_{i,x},\ \tilde{f}_{i,x}$	A release time and a completion time of j_i, respectively, and the start time and the failure time of j_i at r_x, respectively
$\eta(\tilde{f}_{i,x})$	A failure probability at the time $\tilde{f}_{i,x}$
$s_i \subseteq S_\lambda,\ S^{opt}$	ith sub-schedule of schedule λ and an optimal schedule
$\mathcal{S},\ \mathcal{A}_i$	A set of all possible schedules and a set of allocated resources to j_i
$J_i,\ j_i^k \subseteq J_i$	A job chain of j_i and kth backup copy of j_i
$\pi_i,\ r(j_i^k),\ T(j_i^k)$	A policy commonly adopted by $\mathcal{A}_i$, a resource scheduled for j_i^k, and a time slot assigned to j_i^k
$Q_\lambda^{sch},\ Z(\cdot)$	The quality of a given λth schedule and a normalization function
$\epsilon(\cdot), \zeta_\lambda,\ \sigma(\zeta_\lambda)$	A penalty coefficient, a job failure rate of S_λ, and the standard deviation of ζ_λ
$g \leqslant g_{max}$	The number of generations
$P_g,\ ch_u \subseteq P_g,\ ch^{opt}$	gth population, uth chromosome, and an optimal chromosome
$f_u,\ \mathcal{P},\ \mathcal{F}$	uth fitness, a population size, and the average number of failures
$j_{i,x}^k,\ \tilde{s}_{i,x}^k,\ \tilde{c}_{i,x}^k,\ \tilde{f}_{i,x}^k$	A backup copy j_i^k allocated to r_x and a start time, a completion time, and a failure time of $j_{i,y}^k$, respectively
$c(j_{i,x}^k),\ p(j_{i,x}^k)$	A completion state of $j_{i,x}^k$ and a rescheduling policy mode for $j_{i,x}^k$
$D_{i,x\to y}^k$	The data transferring delay of j_i between r_x and r_y
$v_i^k,\ \rho_{x,y}^*,\ \rho_i,\ \mu_i$	A data size of j_i^k and an adjusted transmission rate, a transmission rate, and a failure rate of j_i^k on links between r_x and r_y, respectively
$O_{i,x}^k,\ n_i^k,\ o_x,\ \gamma_{i,y}^{k+1}$	State-tracking overhead of $j_{i,x}^k$, the number of state traces for $j_{i,x}^k$, a unit overhead for checkpointing at r_x, and a remaining length of $j_{i,y}^k$
τ_x	A period of time spent by each checkpointing operation on r_x
$\varphi(\tilde{s}_{i,x}^k), \eta(\tilde{f}_{i,x}^k)$	A probability of continuously processing $j_{i,x}^k$ after $\tilde{s}_{i,x}^k$ and a probability of job failure at $\tilde{f}_{i,x}^k$

sequential batch jobs (i.e., bag-of-tasks) with loosely-coupled intercommunication [10]. Due to the properties of those jobs, they are assumed to be computationally intensive and non-preemptable [15].

Definition 1. Let $R_c = \sum_{x=1}^{m} c_x$ denote the total number of reference computing resources which are given within Δt. The workload intensity at time Δt imposed on the resources, $v(\Delta t)$, can be expressed as

$$v(\Delta t) \simeq \sum_{i=1}^{N} \frac{l_i}{R_c \cdot \Delta t}. \quad (1)$$

Intuitively, $v(\Delta t)$ is calculated by dividing the length of jobs arriving in Δt by the amount of resources that are able to process jobs. The successful completion of a job is strongly correlated with its length and a degree of consistent availability within the given time periods.

2.2. Resource failure

As analyzed in [3], the Weibull distribution is commonly used to model the time until a resource failure of various large scale DCS. Thus, our scheduling system is assumed to follow the Weibull failure law of random variable x, $h(x) = (\beta/\alpha) \cdot (x/\alpha)^{\beta-1}$ as the hazard function, where $\beta > 0$ and $\alpha > 0$ represent the shape parameter and the scale parameter, respectively. We can determine the overall level of system failure by adjusting values of two parameters as discussed in [16].

Definition 2. The failure rate of r_x during the execution of j_i can be expressed as $\eta(\tilde{f}_{i,x}) = Pr[\tilde{s}_{i,x} < \tilde{f}_{i,x} < \tilde{s}_{i,x} + \tilde{l}_{i,x} \mid \tilde{f}_{i,x} > \tilde{s}_{i,x}]$, where $\eta(\tilde{f}_{i,x})$ is the conditional probability function of a resource failure; $\tilde{s}_{i,x}$ and $\tilde{f}_{i,x}$ represent the start time and the failure time of j_i at r_x, respectively. This function can be then transformed to $\{F(\tilde{s}_{i,x} + \tilde{l}_{i,x}) - F(\tilde{s}_{i,x})\}/R(\tilde{f}_{i,x})$, where $F(\cdot)$ and $R(\cdot)$ are the cumulative distribution function and the reliability function $(1 - F(x))$, respectively, of the Weibull distribution model. Additionally, failures of resource are assumed to be statistically independent and we suppose that the job scheduler becomes aware of the failures within a negligible amount of time.

2.3. Heterogeneous rescheduling policy

For the fault tolerance[1] of job scheduling, we consider that each job would be immediately rescheduled by one of following static fault-tolerant policies $\mathbf{\Pi}$, when a resource failure occurs. Hence, each r_x is characterized by $c_x,\ \eta(\tilde{f}_{i,x})$, and $\Pi_x \subset \mathbf{\Pi}$, where Π_x denotes rescheduling policies supported by r_x. These factors directly correspond to job execution delay in the time-dependent scheduling problem. With $\mathcal{J}$, this type of reconfiguration should be planned in a batch schedule prior to the job assignment in order to improve flexibility in batch scheduling. As a result, a new scheduling scheme is required in order to provide the ability to deal with large number of possible combinations of $\mathcal{J},\ \mathcal{R}$, and $\mathbf{\Pi}$.

- Immediate restart policy (IRP): Non-checkpointable applications must restart from the beginning when a failure occurs. In this strategy, a job scheduler immediately restarts a job at the same resource after the resource availability is restored to required levels [17,18].

[1] Fault tolerance is a robustness index of rescheduling decisions that can immediately response to an unexpected resource failure with the least deadline violations. This measure is quantitatively calculated as the average job failure rate in this paper.

- Stateless migration policy (SLMP): This conventional migration scheme [19,20] is also designed for the non-checkpointable jobs. If failure is detected, then a migrant job is reactivated with relevant data transferred from the first resource to the backup resource. So, delays are inevitable due to data movement and job re-execution.
- State-tracking migration policy (STMP): In contrast to IRP and SLMP, this rescheduling strategy uses a checkpointing technique [21,22] as a fast recovery method for abnormal job behaviors. Thus, a backup resource reads the previously saved transient states of a migrant job from the state-tracking server. A job scheduler can then restart the job from the failed point.
- Replicated execution policy (REP): Job replication involves duplicating the same job for different resources for performing multiple and simultaneous executions [16,23]. This delay-tolerant strategy can be used to guarantee the assured completion of critical missions which necessarily require hard-type constraints.

As our preliminary results shown in Fig. 1, some policies have held a dominant position in one of two metrics, with the expected failure probability varying for all jobs, $E[\eta]$. In case of $E[\eta] \leqslant 0.2$, it is hard to determine which policy is generally superior to the others even with two objectives, so that an optimal solution is rarely achieved if multiple conflict objectives are considered. On the other hand, REP and STMP could be better approaches to cope with resource failures, when $E[\eta]$ gets closer to 0.7. However, the costs for occupying redundant backup resources remain uncertain and results are variable even under the similar value of $E[\eta]$. Nevertheless, the existing fault-tolerant scheduling schemes dependent on one static policy have paid less attention to the different aspects of other policies. These technical issues present significant challenges to a scheduling scheme and have motivated us to formulate MPDP and to propose a promising mGA-based MJS scheme.

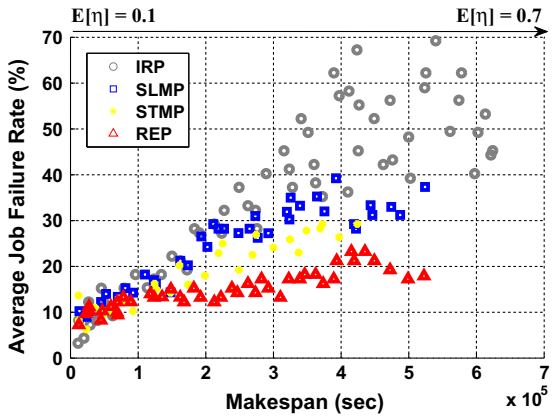


Fig. 1. Comparison of fault tolerance levels of different rescheduling policies with variation of $V(\Delta t)$ in homogeneous policy based DCS; the average job failure rate is calculated as $\frac{\text{number of failed jobs}}{\text{number of all jobs}} \times 100(\%)$ and the makespan is defined as $max\{\tilde{c}_i | \forall i \subseteq \mathcal{J}\}$, where $\tilde{c}_i$ denotes the completion time of j_i.

3. Problem definitions

3.1. Multihybrid policy decision problem

In a fault-tolerant job scheduling context, a multihybrid policy decision problem (MPDP) can be defined as finding the optimal (most feasible) schedule S^{opt} for all batch jobs $\mathcal{J}$ under heterogeneous rescheduling policies $\mathbf{\Pi}$. That is,

$$s_i : j_i \mapsto \mathcal{A}_i, \ \forall s_i \subseteq S_\lambda,$$
$$\mathcal{A}_i = \{r_x, r_k, \ldots, r_l\}, \ \mathcal{A}_i \subset \mathcal{R},$$

where $\mathcal{A}_i$ denotes a set of allocated resources to j_i. The assignment of j_i to $\mathcal{A}_i$ is represented by the mapping symbol $\mapsto$ according to the ith sub-schedule s_i, which is an element of an arbitrary schedule λ, S_λ. Based on the primary-backup fault tolerance model [24], if j_i is assigned to more than one resource, then j_i is classified into two classes: primary copy j_i^1 and backup copies j_i^k, $k = 2, 3, \ldots, |\mathcal{A}_i|$. Those copies for j_i belong to a job chain i, J_i and are scheduled by a single policy π_i that is commonly adopted by $\mathcal{A}_i$ as $\pi_i = \cap_{r_p \subseteq \mathcal{A}_i} \Pi_p$, $\pi_i \subseteq \mathbf{\Pi}$.

Remark 1. This means that the quality of scheduling jobs dynamically varies according to the combinations of $\mathcal{J}$, $\mathcal{R}$, and $\mathbf{\Pi}$. Thus, the MPDP is considered as a combinatorial optimization problem.

In the MPDP, $S_\lambda \subset \mathcal{S}$ must satisfy the following two constraints with respect to the exclusive use of resources among job copies in order to guarantee the system fault tolerance, where $\mathcal{S}$ denotes the solution space. Let $r(j_i^k)$ denote a resource that is scheduled for j_i^k and let $T(j_i^k)$ denote a time slot assigned to j_i^k.

$$C1.\ \forall j_i \subseteq \mathcal{J}, \ r(j_i^k) \neq r(j_i^q), \ k \neq q,$$
$$C2.\ r(j_i^k) = r(j_u^c), \ T(j_i^k) \cap T(j_u^c) = \emptyset.$$

The constraints regarding control dependency, in turn, ensure the following: (C1) A system tolerates the job abortion caused by a resource failure, if and only if, the precedent copy j_i^k and the backup copy j_i^q of j_i are respectively allocated to two different resources, when a failure occurs. (C2) This constraint implies that overlapped execution is not allowed for two different copies since only one job can be assigned to a resource at an instance of time.

Remark 2. MPDP with the above instance is strongly NP-complete because the volume of its search space to be explored is $O(N!M^N)$, even though $|\mathbf{\Pi}| = 1$ and a failure does not occur. It is hard to confirm whether an optimal solution $S^{opt} \subset \mathcal{S}$ is achieved by a deterministic approach (e.g., an approximate algorithm) in the MPDP.

3.2. Optimal schedule for MPDP

The goal is to determine S^{opt} delivering mutually and fairly optimized performance in three respects: makespan, load balance, and robustness. To assess the quality of a

given λth schedule, Q_λ^{sch}, we need to transform different quantities of the first two objectives into the same domain in which they are normalized and are then combined by the weighted sum method as follows:

$$Q_\lambda^{sch} = \delta\left(\frac{m_{max} - m_\lambda}{m_{max} - m_{min}}\right) + \omega\left(\frac{u_\lambda - u_{\min}}{u_{max} - u_{min}}\right), \quad (2)$$

where m_λ and u_λ denote the makespan and the standard deviation of utilization of S_λ. Also, m_{max} (u_{max}) and m_{min} (u_{min}) specify the maximum value and the minimum value of makespan (average utilization), respectively. To compute m_{max}, m_{min}, u_{max}, and u_{min}, we solve single-objective optimization problems with a similar configuration of $\mathcal{J}$, $\mathcal{R}$, and $\mathbf{\Pi}$ by maximizing and minimizing the corresponding objective. Weights $\delta \subseteq [0,1]$ and $\omega \subseteq [0,1]$ correlate to their importance respectively, where $\delta + \omega = 1$. For gradual changes in the weights, we also set δ to $|\sin(2\pi t/f)|$, where t is the time period and f is the change frequency of δ.

The larger Q_λ^{sch} guarantees the better scheduling performance. Due to the conflicting nature of the objectives, different schedules produce trade-offs between m_λ and u_λ, in which the upper bound of one is relaxed and the other is re-optimized for each generation of the optimization process. As such, there could be a set of multiple feasible solutions in each stage of the optimization process. To exclude the dominated solutions, we further penalize inferior solutions regarding a job failure rate ζ_λ by reducing their Q_λ^{sch} values in proportion to their degrees of the following constraint violation:

$$C3.\ \tilde{r}_i \leqslant \tilde{s}_i^1 < \tilde{c}_i^{|\mathcal{A}_i|} \leqslant \tilde{d}_i,$$

where $\tilde{r}_i$, $\tilde{s}_i^1$, and $\tilde{c}_i^{|\mathcal{A}_i|}$ denote the release time of j_i, the start time of j_i^1, and the completion time of $j_i^{|\mathcal{A}_i|}$, respectively. For $\forall j_i^k \subseteq J_i$, $\tilde{c}_i^{|\mathcal{A}_i|} = E[max\{\tilde{c}_i^k\}] > \tilde{d}_i$ means that the execution of j_i is failed. However, counting the failures gives unfair importance to the jobs unless the jobs are of equal criticality, so that the standard deviation of ζ_λ, $\sigma(\zeta_\lambda)$, could be a better measure of how unfair S_λ is to $\forall j_i \subseteq \mathcal{J}$. Let $Z(\cdot) \subseteq [0,1]$ denote a normalization function for m_λ and u_λ. Then, the reformulated quality assessment function is given by

$$Q_\lambda^{sch} = (\delta \cdot Z(m_\lambda) + (1-\delta) \cdot Z(u_\lambda)) + (\epsilon(g) - \sigma(\zeta_\lambda)). \quad (3)$$

In Eq. (3), $\epsilon(g)$ represents a penalty coefficient for S_λ provided in the gth generation and is assumed to be dynamically adjustable for each generation. The level of $\epsilon(g)$ is controlled as

$$\epsilon(g) = \begin{cases} \min(\sigma(\hat{\zeta}_\theta)) & \text{if } g = 0, \\ \epsilon(0)\left(1 - \frac{g}{g_{max}}\right) & \text{if } 1 \leqslant g < g_{max}, \\ 0 & \text{if } g \geqslant g_{max}, \end{cases} \quad (4)$$

where $\sigma(\hat{\zeta}_\theta)$ is the standard deviation of failure rate of the top θth solution and θ is set to 20% of initial solutions. Once $\epsilon(0)$ is set, $\epsilon(g)$ decreases in proportional to $\left(1 - \frac{g}{g_{max}}\right)$, when g gets closer to g_{max}. For the higher g, the smaller $\epsilon(g)$ would be applied to evaluate the quality of solutions. If any solution fails to evolve in a better direction, its quality is assessed to be degraded. In case of $g \geqslant g_{max}$, the optimization process stops; thus, $\epsilon(g) = 0$. This ϵ-level controlling method does not require knowing a priori the preference of ζ_λ and can give Q_λ^{sch} a reward in case of $\sigma(\zeta_\lambda) < \epsilon(g)$.

Lemma 1. *With a systematic variation of $\epsilon(g)$, two feasible solutions such that $Q_i^{sch} \approx Q_j^{sch}$ $(S_i, S_j \subseteq \mathcal{S}_f)$ can be compared in terms of a robustness metric, where every schedule in $\mathcal{S}_f$ satisfies C2 and C3.*

Proof. Although the optimal solutions of MPDP provide the points (m_λ, u_λ) of the Pareto optimal solutions,[2] some of them may not satisfy C3 to a certain extent. In this case, Eq. (4) gives us a way of measuring a degree of constraint (C3) variation. Specifically, this method helps an optimization process select a particular solution based on additional preference information, $\sigma(\zeta_\lambda)$ about the two objectives by varying the value of $\epsilon(g)$ systematically. Hence, a set of mostly non-dominated solutions S^{opt} can be obtained with respect to the three metrics of makespan, load balance, and failure rate. □

4. The proposed job scheduling scheme for MPDP

The purpose of a multihybrid job scheduling (MJS) scheme is to confidently guarantee S^{opt} for the problem MPDP. Due to its NP-completeness as discussed in Remark 2, we first construct the MJS scheme, according to the stochastic search operations of mGA, as described in Algorithm 1. However, particular logic, such as initialization and evaluation, strongly depend on what problem should be solved. On account of the inherent specialty of MPDP, we then propose two processes for solution representation and realization. The former codifies a schedule to MPDP and the latter computes sequences and time slots of each job, in order to make mGA compatible with MPDP. One key factor to consider is that these processes must not contribute to the increase of the computational complexity of the MJS scheme.

4.1. Solution representation process

A schedule conforming with C1, C2, and C3, $S_\lambda \subset \mathcal{S}$, (termed a chromosome in mGA) has a unique combination of three factors, $\mathcal{J}$, $\mathcal{R}$, and $\mathbf{\Pi}$ in MPDP. To codify a solution to MPDP, we define a new real-coded encoding schema $\acute{E}_{rc}$, in which $s_i \subseteq S_\lambda$ is represented by 2-tuples referred to as a gene, such as (*job index i*, *resource index x*); hence, S_λ is a set of randomly generated pairs. For ease of demonstrating $\acute{E}_{rc}$, we assume that a policy commonly adopted by a set of some resources is known and for example, a given chromosome λ, ch_λ, is easily translated into S_λ as shown in Fig. 2.

[2] $S_i \subseteq \mathcal{S}_f$ is said to be Pareto optimal if and only if $S_j \subseteq \mathcal{S}_f$ such that $\forall S_i, S_j$ $(i \neq j), Z(m_j) = Z(m_i)$ with strict inequality under at least one condition, $Z(u_j) < Z(u_i)$.

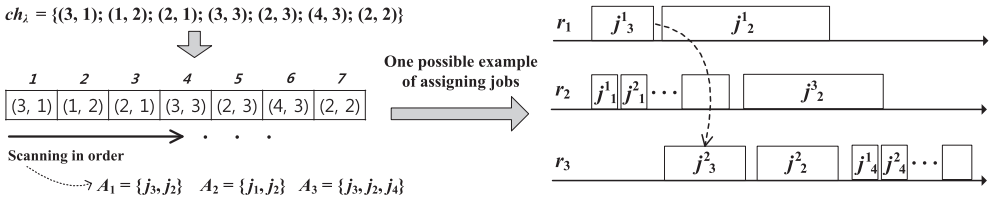


Fig. 2. An example of translating ch_λ into S_λ in the Gantt chart.

In ch_λ, j^1_3 is allocated earlier to r_1 than j^1_2 because the execution order of jobs is read by the order of appearance in ch_λ and j^2_3 is also migrated to r_3 if its π_3 is SLMP or STMP. Since j_1 and j_4 are allocated to one resource, they will be re-executed by IRP if a failure occurs. Moreover, j^1_2 is secondly scheduled on r_1 and then its copies j^2_2 and j^3_2 will be run in parallel if its π_2 is REP.

4.2. mGA-based MJS operations for solution reproduction

Algorithm 1 illustrates that mGA evolves an initial population of possible solutions to MPDP being solved by means of reproduction operations under the principle of the survival of the fittest. We briefly explain each phase of the proposed mGA-based MJS scheme.

Algorithm 1. mGA-based MJS operations

Require: $\forall j_i \subseteq \mathcal{J},\ l_i,\ 1 \leqslant i \leqslant N,\ \mathcal{R},\ c_x,\ 1 \leqslant x \leqslant M,$
$\mathbf{\Pi},\ \eta(\cdot),\ g = 0$
Generate P_g according to $\acute{E}_{rc}$ while abiding with C1
while $g \leqslant g_{max}$ **do**
f_u = *Evaluate* (ch_u), $\forall ch_u \subseteq P_g,\ 1 \leqslant u \leqslant U$;
If $\exists ch_u \subseteq P_g$ satisfying that $f_u \geqslant f_{th}$ **then**
$ch^{opt} \leftarrow ch_u$ and break;
end if
$(ch_a, ch_b) \leftarrow$ Select parents by $Pr_{sel}(ch_u)$;
$ch_o \leftarrow$ Cut-and-Splice(ch_a, ch_b);
$\hat{ch}_o \leftarrow$ Mutate(ch_o) and then add $\hat{ch}_o$ in P_{g++};
end while
If $g \geqslant g_{max}$ **then**
$ch^{opt} \leftarrow ch_u$ such that $max\{f_u, u \subseteq [1, U]\}$;
end if
Output: Translate ch^{opt} into S^{opt};
Start to dispatch $\mathcal{J}$ according to S^{opt};

(i) *Initialization*. Based on $\acute{E}_{rc}$, mGA starts with a set of contending trial solutions, the initial population $P_g(g = 0)$, which is comprised of a fixed number of $ch_u,\ 1 \leqslant u \leqslant U$, where U is a fixed integer number.
(ii) *Evaluation*. To evaluate a fitness value for each ch_u, f_u, *Evaluate*(ch_u) (described in Algorithm 2) is called in order to obtain f_u. If f_u is sufficiently large enough compared to a predefined threshold f_{th}, then ch_u becomes an optimal allocation ch^{opt} and this procedure is finally terminated.
(iii) *Selection*. Otherwise, chromosomes having relatively higher fitness values should be selected by the probability $Pr_{sel}(ch_u) = \left(f_u \Big/ \sum_{u=1}^{U} f_u\right)$ in a roulette-wheel selection [12] to preserve superior solutions, which would be evolved in the next generation.
(iv) *Recombination*. To obtain globally evolved solutions, cut-and-splice swaps the first d genes between ch_a and ch_b, where d is a positive integer, $1 \leqslant a,\ b \leqslant U$ and $a \neq b$, producing offspring ch_o. To explore local solutions, mutation is then performed by randomly exchanging two genes within each ch_o. After these two operations, we can have more feasible offspring $\hat{ch}_o$ in the next generation P_{g++}.
(v) *Termination*. All phases are iterated until $f_u \geqslant f_{th}$ or $g \geqslant g_{max}$ is satisfied. All of the operations must abide with the two constraints (C1 and C2).

The mGA-based MJS scheme is described by a stochastic process that evolves over time in a probabilistic manner to search an optimal state, ch^{opt}. Since its intermediate transitions do not depend on the time, they have Markov properties and are homogeneous [25].

Lemma 2. *We consider that the population size, $\mathcal{P}$ and the cardinality of the state space, $|\mathcal{S}|^{\mathcal{P}}$ are finite. Assuming that the above operations are treated as a random transition from $\mathcal{S}^{\mathcal{P}}$ to $\mathcal{S}^{\mathcal{P}}$, Algorithm 1 finally converges to ch^{opt} regardless of the search space.*

Proof. We will prove Lemma 2 by establishing the connection between the limiting behavior of Markov chains and the stochastic convergence of a random population. Through finite Markov chains, when $g \to \infty$, its state transition matrix P_{st} converges to a unique stable matrix P^{∞}_{st}, which is the product of state transition matrices caused by the operations, regardless of the initial distribution of the population, p^0 (stationary). For arbitrary p^0, there is a unique limiting distribution, $p^{\infty} = \lim_{g\to\infty} p^0 P^{\infty}_{st} = (p^{\infty}_1, p^{\infty}_2, \ldots, p^{\infty}\mathcal{S}^{\mathcal{P}}, 0, \ldots, 0)$, where p^{∞}_i is the probability that the limiting state is at ith state of $\mathcal{S}^{\mathcal{P}}$. Therefore, the probability of the limiting state being in global optimal states is obviously one, i.e., $\sum_{i=1}^{|\mathcal{S}|^{\mathcal{P}}} p^{\infty}_i = 1$. This means that there must be at least one ch^{opt}, when g $\to \infty$. □

Remark 3. mGA generally satisfies the following property: $Pr[ch_{u+1} \not\subseteq P^{opt} | ch_u \subseteq P^{opt}] = 0 (\forall u = 0, 1, \ldots)$, where P^{opt} stands for a population containing at least one ch^{opt}.

This property implies that mGA never loses ch^{opt} once it has been found during the evolution of a population. Thus, mGA can be modeled by an absorbing Markov chain.

4.3. Solution realization process

Although, in practice, computation times are not deterministic as discussed in [1], each ch_u needs to be identified in terms of the (*i*) sequences and (*ii*) execution times of job copies in order to evaluate (*iii*) its quality. This fact further requires a new approach for solution realization (i.e., decoding and evaluation). In the proposed mGA-based MJS scheme, each of the above factors is achieved by the three functions of *decideSequence*(), *estimateProbExeTime*(), and *assessSchedQuality*(), as described in Algorithm 2.

4.3.1. decideSequence() under precedence constraints

In the example of ch_u addressed in Section 4.1, j_3^2 would be declaimed at r_3 and then a state of $T(j_3^2)$ would be updated as idle, if j_3^1 is successfully completed (passive rescheduling). On the other hand, j_2^2 and j_2^3 should be invoked regardless of j_2^1's execution result (active rescheduling). As it stands, invocations of backup copies depend upon the success or failure of their direct predecessor and the commonly adopted rescheduling policy due to the precedence constraints, which define a partial order between job copies in the same job chain. For clearly indicating which resource is allocated to a particular job copy, hereafter we add the subscript x to j_i^k as $j_{i,x}^k$; however, we often drop the argument of job copy when it is clear from the context.

Algorithm 2. Evaluate(ch_u)

Require: $\forall ch_u \subseteq P_g,\ 1 \leqslant u \leqslant U$;
  for $\exists ch_u \subseteq P_g$ **do**
    decideSequence();
    estimateProbExeTime();
    $Q_u^{sch} \leftarrow$ *assessSchedQuality*();
  end for
Output: Return Q_u^{sch};

Definition 3. If j_u^w is scheduled immediately prior to j_i^k on the same resource and is associated with J_u other than J_i, it is said to be an adjacent copy of j_i^k.

Lemma 3. *Let $c(j_{i,x}^k)$ and $p(j_{i,x}^k)$ denote a completion state of $j_{i,x}^k$ and a policy mode adopted for reallocating $j_{i,x}^k$, respectively, where $c(j_{i,x}^k)$ is either success (1) or failure (0); and $p(j_{i,x}^k)$ is either active (1) or passive (0). The start time of arbitrary j_i^k scheduled on r_y (denoted by $\tilde{s}_{i,y}^k$) can be achieved by considering $c(j_{i,x}^{k-1})$ and $p(j_{i,y}^k)$, where $x \neq y$ as*

$$\tilde{s}_{i,y}^k = \begin{cases} null & \text{if } c(j_{i,x}^{k-1}) = 1 \wedge p(j_{i,y}^k) = 0, \\ max(\tilde{f}_{i,x}^{k-1}, \tilde{f}_{u,y}^w) & \text{if } c(j_{i,x}^{k-1}) = 0 \wedge p(j_{i,y}^k) = 0, \\ min(\tilde{r}_{i,y}^k, \tilde{f}_{u,y}^w) & \text{otherwise.} \end{cases} \quad (5)$$

Proof. We prove Lemma 3 by introducing precedence constraints between $j_{i,y}^k$, its direct predecessor $j_{i,x}^{k-1}$, and its adjacent copy j_u^w with three cases. (a) If $j_{i,x}^{k-1}$ finishes without any failure and rescheduling follows a principle of linear job allocation, $j_{i,y}^k$ would be de-allocated; thus, $\tilde{s}_{i,y}^k$ is null. (b) If $j_{i,x}^{k-1}$ fails to complete its execution under a passive policy such as IRP, STMP, and SLMP, j_i^k must be released on r_y after its predecessor's failure time $\tilde{f}_{i,x}^{k-1}$. Also, there may be j_u^w. In this case, $\tilde{s}_{i,y}^k$ would be set immediately after its adjacent copy's failure time $\tilde{f}_{u,y}^w$ to avoid the overlapped run. (c) For an active policy (i.e., REP), $j_{i,y}^k$ has priority over $j_{u,y}^w$, if the scheduled allocation of $j_{i,y}^k$ is overlapped with that of $j_{u,y}^w$ $(\tilde{s}_{u,y}^w < \tilde{s}_{i,y}^k < \tilde{f}_{u,y}^w)$. Otherwise, $j_{i,y}^k$ can be started immediately after its release time $\tilde{r}_{i,y}^k$. □

Therefore, Lemma 3 implies that in ch_u, $\tilde{c}_{i,y}^k$ is also calculated by adding $\tilde{s}_{i,y}^k$ to its execution time, so that $T(j_i^k)$ on r_y can be realized. The time complexity of *decideSequence*() is bound to $O(\mathcal{P}MN\mathcal{F}\log N\mathcal{F}) \approx O(MN\log N)$ in the evaluation phase, where N and $\mathcal{F}$ denote the total number of jobs and the average number of failures for each job, respectively, with $\mathcal{P}$ and $\mathcal{F}$ as constants. In addition, $O(N^2M)$ is the worst case complexity of this sub-algorithm, when $\mathcal{F} \geqslant N$. However, $N \gg \mathcal{F}$ is a more appropriate assumption that can be supported by the realistic workload [14] and the failure trace archives [3]. Hence, the worst case is rarely expected, and essentially meaningless, in practical distributed computing.

4.3.2. estimateProbExeTime() with different policies

We now propose a method for estimating a probabilistic execution time (PET) of each job copy under four different fault-tolerant scheduling policies. In this section, we define $PET^{\pi_i}(i,k,x)$ as a PET of j_i^k that is assigned on r_x by π_i, where it is computed by the sum of the expected execution time $(eet_{i,x}^k)$ and the expected waste time $(ewt_{i,x}^k)$. A PET for j_i is finally obtained as $PET^{\pi_i}(i) = \sum_{r_x \subseteq \mathcal{A}_i} PET^{\pi_i}(i,k,x)$.

(1) IRP – Since we suppose that a failure is transient, IRP uses the random exponential back-off strategy to gracefully back off jobs on resources as deployed in [18]. Retries should occur at intervals that increase exponentially but with some random variation. Fig. 3(a) shows that jobs arriving within a given scheduling cycle time t_{sc} are queued for setup time t_{wq} and then the proposed MJS scheme attempts to re-dispatch j_i^k at the same r_x until a failure no longer exists. By definition of expectation $E[\cdot]$, $eet_{i,x}^k$ and $ewt_{i,x}^k$ can be predicted, respectively, as follows:

$$eet_{i,x}^k = E[\tilde{c}_{i,x}^k - \tilde{f}_{i,x}^{k-1}] = \tilde{l}_{i,x}, \quad (6)$$

$$ewt_{i,x} = \sum_{l=1}^{k-1} ewt_{i,x}^{l} = E[\tilde{f}_{i,x}^{k-1} - \tilde{s}_{i,x}^{1}] = \sum_{l=1}^{k-1} \eta(\tilde{f}_{i,x}^{l}) \cdot awt_{i,x}, \quad (7)$$

where $\tilde{c}_{i,x}^{k}$ and $\tilde{f}_{i,x}^{k-1}$ denote the completion time of j_i^k and the failure time of j_i^{k-1} assigned on r_x, respectively. $\eta(\tilde{f}_{i,x}^{l})$ is a probability that the lth job copy of j_i is failed at a given time $\tilde{f}_{i,x}^{l}$. Suppose that a resource failure is uniformly distributed. The average waste time of $j_{i,x}$, $awt_{i,x}$, can be expressed as $\frac{\tilde{l}_{i,x}}{2}$ because a failure may or may not occur within an assigned time slot. Then, the expected waste time of $j_{i,x}^{l}$, $ewt_{i,x}^{l}$ $(1 \leqslant l < k)$, can be expressed as $\eta(\tilde{f}_{i,x}^{l}) \times \frac{\tilde{l}_{i,x}}{2}$. Eq. (7) thus means the total waste time of j_i due to the $k-1$ failures. Then, we have $PET^{IRP}(i)$ as follows:

$$PET^{IPR}_{r_x \subseteq \mathcal{A}_i}(i,x) = \tilde{l}_{i,x} + \tilde{l}_{i,x} \cdot \frac{\sum_{l=1}^{k-1} \eta(\tilde{f}_{i,x}^{l})}{2} = \frac{l_i}{c_x} \cdot \left(1 + \frac{\sum_{l=1}^{k-1} \eta(\tilde{f}_{i,x}^{l})}{2}\right). \quad (8)$$

(2) STMP – In general, STMP involves two types of migration overheads imposed by data movement and state-tracking, as illustrated in Fig. 3(c), where we assume that j_i is scheduled to be migrated from r_x to r_y.

Data transferring delays. Unlike IRP, the transfer time of job data can be considered the main overhead when a failure occurs. Such a network delay is formulated as

$$D^{k}_{\substack{i,x\to y \\ r_x,r_y \subseteq \mathcal{A}_i \\ x\neq y}} = \begin{cases} 0 & \text{if } r_x = r_{|\mathcal{A}_i|}, \\ \frac{v_i^k}{\rho^*_{x,y}}, \rho^*_{x,y} = \frac{\sum_{i=1}^{p} \mu_i}{\sum_{i=1}^{p} \frac{\rho_i}{\mu_i}} & \text{otherwise.} \end{cases} \quad (9)$$

In Eq. (9), the data transferring delay of j_i between r_x and r_y (denoted by $D^{k}_{i,x\to y}$) is calculated by dividing v_i^k by $\rho^*_{x,y}$, where v_i^k and $\rho^*_{x,y}$ represent the data size of j_i^k and the adjusted transmission rate between r_x and r_y, respectively. $\rho^*_{x,y}$ particularly illustrates the network behavior in terms of the transmission rate ρ_i and the failure rate μ_i of j_i^k on all links from 1 to p in the path between r_x and r_y.

State-tracking overheads. During the job processing, a state-tracking server periodically writes checkpoints and updates states of executing j_i^k on r_x. If a failure is detected, the MJS scheme attempts to reserve a secondary resource r_y from $\mathcal{A}_i$. Then, job and relevant data are transferred to r_y. Finally, r_y reads a previously-saved checkpoint from the state-tracking server to compute a remaining length of j_i^k and then restarts j_i^k at a failed point. An equation for calculating the state-tracking overhead of $j_{i,x}^k$, $O_{i,x}^k$, is derived as

$$O_{i,x}^{k} = n_i^k \cdot o_x, \quad (10)$$

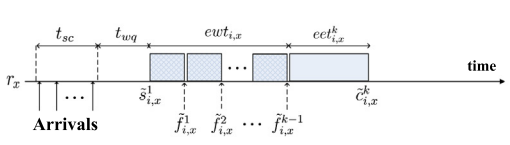
where n_i^k and o_x denote the number of state traces and a unit overhead required for performing one checkpointing operation, respectively. o_x is inversely proportional to c_x and n_i^k is calculated differently in two cases, as given by

$$n_i^k = \begin{cases} \left\lfloor \frac{\tilde{l}_{i,x}^{k}}{\tau_x} \right\rfloor & \text{if } c(j_{i,x}^k) = 1, \\ \left\lfloor \frac{awt_{i,x}^{k}}{\tau_x} \right\rfloor & \text{if } c(j_{i,x}^k) = 0. \end{cases} \quad (11)$$

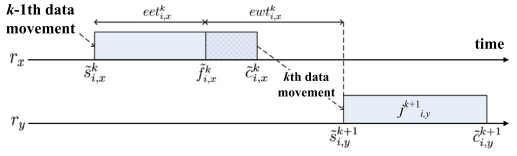
In Eq. (11), τ_x in the denominator represents a period of time spent by each checkpointing operation on r_x and we assume that τ_x is identical to all resources. n_i^k is calculated by dividing $\tilde{l}_{i,x}^{k}$ by τ_x if there is no failure during the execution of $j_{i,x}^k$. Otherwise, $awt_{i,x}^k$ should be considered instead of using $\tilde{l}_{i,x}^{k}$ because a failure causes slowdown of its execution.

For ease of description, hereafter we separately denote the state-tracking overhead of j_i^k failed at r_x as $\acute{O}_{i,x}^k$. The remaining length of $j_{i,y}^k$, $\chi_{i,y}^{k+1}$, is then calculated by subtracting its elapsed time from the original size of j_i, l_i. Thus, $\chi_{i,y}^{k+1}$ is given by

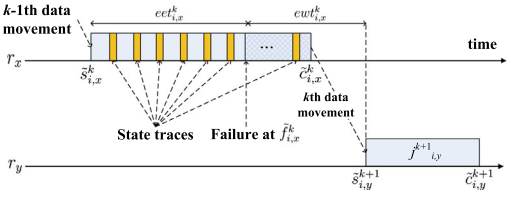
$$\chi_{i,y}^{k+1} = l_i - \left(\left\lfloor \frac{awt_{i,y}^{k}}{\tau_y} \right\rfloor \cdot \tau_y \cdot o_y \right). \quad (12)$$



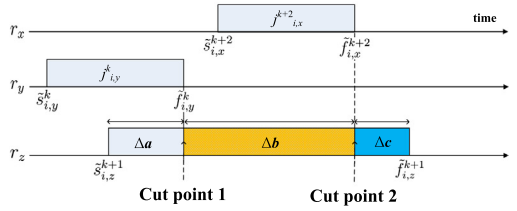
(a) IRP-ruled job copy



(b) SLMP-ruled job copy



(c) STMP-ruled job copy



(d) REP-ruled job copy

Fig. 3. An illustration of rescheduled job copy in (a) IRP, (b) SLMP, (c) STMP, and (d) REP.

From Eqs. (9)–(11) we can derive $eet^k_{i,x}$ and $ewt^k_{i,x}$ as

$$eet^k_{i,x} = E\left[\tilde{f}^k_{i,x} - \tilde{s}^k_{i,x}\right] = \varphi(\tilde{s}^k_{i,x}) \cdot \left((1-\eta(\tilde{f}^k_{i,x})) \cdot (\tilde{l}^k_{i,x} + O^k_{i,x})\right), \quad (13)$$

$$ewt^k_{i,x} = E\left[\tilde{c}^k_{i,x} - \tilde{f}^k_{i,x}\right] = \varphi(\tilde{s}^k_{i,x}) \cdot \left(\eta(\tilde{f}^k_{i,x}) \cdot \left(\frac{\tilde{l}^k_{i,x}}{2} + \acute{O}^k_{i,x} + D^k_{i,x\to y}\right)\right), \quad (14)$$

$$\varphi(\tilde{s}^k_{i,x}) = \begin{cases} 1 & \text{if } k = 1, \\ \prod_{y \subseteq \mathcal{A}_i} \eta(\tilde{f}^{k+1}_{i,y}) & \text{if } \exists j^{k+1}_{i,y} \text{ such that } \tilde{f}^{k+1}_{i,y} \leqslant \tilde{f}^k_{i,x}, \end{cases} \quad (15)$$

where $\varphi(\tilde{s}^k_{i,x})$ and $\eta(\tilde{f}^k_{i,x})$ specify a probability of continuously processing $j^k_{i,x}$ after $\tilde{s}^k_{i,x}$ and a probability of job failure at $\tilde{f}^k_{i,x}$, respectively. As described in Eq. (15), $\varphi(\tilde{s}^k_{i,x})$ is given by the product of the failure probabilities of j^k_i's predecessors. If j^k_i is a primary copy $(k = 1)$, $\varphi(\tilde{s}^k_{i,x})$ is equal to 1.

Therefore, a PET of j_i controlled by STMP is achieved as

$$PET^{STMP}_{r_x, r_y \subseteq \mathcal{A}_i}(i,k,x \to y) = \varphi(\tilde{s}^k_{i,x}) \cdot \left\{\left((1-\eta(\tilde{f}^k_{i,x})\right) \cdot \left(\frac{\chi^k_{i,x}}{c_x} + \left\lfloor \frac{\chi^k_{i,x}}{c_x} \cdot \frac{1}{\tau_x} \right\rfloor \cdot o_x\right) + \eta(\tilde{f}^k_{i,x}) \cdot \left(\frac{\chi^k_{i,x}}{2c_x} + \left\lfloor \frac{\chi^k_{i,x}}{c_x} \cdot \frac{1}{\tau_x} \right\rfloor \cdot o_x + \frac{v^k_i}{\rho^*_{x,y}}\right)\right\}. \quad (16)$$

We note that, with the exception of state-tracking overheads, the PET of a SLMP-ruled job copy (see Fig. 3(b)) can be calculated in a similar manner to STMP.

(3) REP – This replication strategy simultaneously distributes workloads to assigned resources. Fig. 3(d) shows that three redundant job copies (i.e., replicas) are executed on r_x, r_y, and r_z in order to completely finish at least one. $j^k_{i,y}$ starts running at $\tilde{s}^k_{i,y}$, then r_z also has to process $j^{k+1}_{i,z}$ at $\tilde{s}^{k+1}_{i,z}$ because $j^k_{i,y}$ may be failed at any time. If $j^k_{i,y}$ is abnormally aborted at $\tilde{f}^k_{i,y}$ then $j^{k+2}_{i,x}$ subsequently has to be executed from $\tilde{s}^{k+2}_{i,x}$. However, if we assume that $j^{k+2}_{i,x}$ fails at $\tilde{f}^{k+2}_{i,x}$ as well, then $j^{k+1}_{i,z}$ must proceed. In the end, only $j^{k+1}_{i,z}$ is successfully completed at $\tilde{f}^{k+1}_{i,z}$.

Replicas. Due to the parallel precedence constraints, we classify these job copies into two types, a normal replica and a cut replica. Specifically, an arbitrary job copy $j^w_{i,q}$ in J_i can be recognized as a cut replica if the following condition is satisfied: $\tilde{s}^k_{i,p} \leqslant \tilde{f}^w_{i,q} < \tilde{f}^k_{i,p}$, where $k \geqslant 1$, $w \leqslant |\mathcal{A}_i|$, r_p, $r_q \subseteq \mathcal{A}_i$, and $p \neq q$. For instance, $j^{k+1}_{i,z}$ is cut twice by the other replicas at $\tilde{f}^k_{i,y}$ and $\tilde{f}^{k+2}_{i,x}$, respectively; so, $j^{k+1}_{i,z}$ is a cut replica and the others are normal ones. Calculating a PET of $j^{k+1}_{i,z}$ requires a different technique, while PETs for $j^{k+2}_{i,x}$ and $j^k_{i,y}$ can be derived in the same way as estimating PETs under the other policies. According to the two cut points, $eet^{k+1}_{i,z}$ is divided into three parts such as Δa, Δb, and Δc:

$$\begin{aligned} E[\tilde{f}^{k+1}_{i,z} - \tilde{s}^{k+1}_{i,z}] &= \Delta a + \Delta b + \Delta c \\ &= \varphi(\tilde{s}^{k+1}_{i,z})(\tilde{f}^k_{i,y} - \tilde{s}^{k+1}_{i,z}) + \varphi(\tilde{f}^k_{i,y})(\tilde{f}^{k+2}_{i,x} - \tilde{f}^k_{i,y}) + \varphi(\tilde{f}^{k+2}_{i,x})(\tilde{f}^{k+1}_{i,z} - \tilde{f}^{k+2}_{i,x}) \\ &= (\tilde{f}^k_{i,y} - \tilde{s}^{k+1}_{i,z}) + \eta(\tilde{f}^k_{i,y})(\tilde{f}^{k+2}_{i,x} - \tilde{f}^k_{i,y}) + \eta(\tilde{f}^k_{i,y})\eta(\tilde{f}^{k+2}_{i,x})(\tilde{f}^{k+1}_{i,z} - \tilde{f}^{k+2}_{i,x}). \end{aligned} \quad (17)$$

Assuming that the maximum number of failures is set to 3 for each job copy, we then generalize the PET estimation formula for a cut replica $(j^{k+1}_{i,z})$ as follows:

$$PET^{REP}_{r_z \subseteq \mathcal{A}_i}(i,k+1,z) = \varphi(\tilde{s}^{k+1}_{i,z})(\tilde{f}_{i,u} - \tilde{s}^{k+1}_{i,z}) + \sum_{\substack{x,y \subseteq \mathcal{A}_i, 1 \leqslant r_x, r_y \leqslant |\mathcal{A}_i| \\ \tilde{s}^{k+1}_{i,z} \leqslant \tilde{f}^k_{i,y} < \tilde{f}^{k+2}_{i,x} \leqslant \tilde{f}^{k+1}_{i,z} \\ \tilde{f}^k_{1,y} \leqslant \tilde{f}^w_{i,r} < \tilde{f}^{k+2}_{i,x}, \neg\exists \tilde{f}^w_{i,r}}} \varphi(\tilde{f}^k_{i,y})(\tilde{f}^{k+2}_{i,x} - \tilde{f}^k_{i,y}),$$
$$\text{where } \tilde{s}^{k+1}_{i,z} < \tilde{f}_{i,u}, \tilde{s}^{k+1}_{i,z} \leqslant \tilde{f}_{i,v} < \tilde{f}_{i,u}, \neg\exists \tilde{f}_{i,v}, \quad \varphi(\tilde{f}^k_{i,y}) = \prod_{r_y \subseteq \mathcal{A}_i, \tilde{f}^k_{i,y} \leqslant \tilde{s}^{k+1}_{i,z}} \eta(\tilde{f}^k_{i,y}). \quad (18)$$

4.3.3. assessSchedQuality() for fitness comparison

The third function of the evaluation phase uses Eq. (3) in order to assess the quality of each $ch_u \subseteq P_g$, $1 \leqslant g \leqslant g_{max}$, where the three objectives such as m_u, u_u, and $\sigma(\zeta_u)$ should be calculated from each ch_u. By iteratively estimating each copy's PET in ch_u, the completion time of r_x, C_x, is calculated by $\sum_{r(j^k_i)=r_x} PET^{\pi_i}(i,k,x)$. Then, m_u is obtained by $max\{C_x | r_x \subseteq \mathcal{A}_i\}$ and the average utilization of each resource is equal to $\sum_{r_x \subseteq \mathcal{A}_i} C_x/(m_u \cdot |\mathcal{A}_i|)$. Further, $\sigma(\zeta_u)$ can be derived by definition of standard deviation. Therefore, by using Eq. (3), Q^{sch}_u can be evaluated. Then, ch^{opt} is finally achieved by $ch^{opt} \subseteq P_g \leftarrow ch_u$ such that $max\{f_u, u \subseteq [1,U]\}$ when $f_u \geqslant f_{th}$ or $g \geqslant g_{max}$.

5. Analysis of the proposed job scheduling scheme

Now we turn our attention to the algorithmic searching performance of the proposed scheme relative to its computational complexity. Since the computational complexity of the proposed scheme is a property of both mGA (search operations) and MPDP (solution realization process) in practice, rather than identifying that of mGA only, the computational time requirement of the former can be calculated as below.

- In the population initialization phase, a solution has $N(\mathcal{F}+1)$ genes, where $\mathcal{F}$ is an integer; it is assumed to be the same for $\forall j \subseteq \mathcal{J}$. Each gene is randomly selected from possible job and resource pairs; its resulting complexity is $O(\mathcal{P}NM(\mathcal{F}+1))$. Since $\mathcal{P}$ and $\mathcal{F}$ are both treated as constants, this operation has complexity $O(NM)$. The roulette-wheel selection sorts individual solutions in descending order by their quality in each generation, so that the complexity is $O(g_{max}\mathcal{P}\log(\mathcal{P}))$. g_{max} and $\mathcal{P}$ are set as constants, respectively. Thus, the selection operation has complexity $O(1)$.
- The cut-and-splice operator is used $\frac{p_c\mathcal{P}}{2}$ times, where p_c is the probability of choosing d $(0 < d \leqslant N(\mathcal{F}+1))$ genes in each solution as discussed. However, p_c and $\mathcal{P}$ are all constants. Therefore, its complexity is determined by the length of a solution, $N(\mathcal{F}+1)$ because this

operator needs to read all genes in each solution. This operator has complexity $O(N)$. The mutation is invoked $p_m\mathcal{P}$ times, where p_m is a probability of randomly visiting two genes in each solution and both p_m and $\mathcal{P}$ are constants. Thus, its complexity is bound as $O(1)$.

Second, we also consider the proposed solution realization process that consists of the three functions for quality evaluation incorporated into the mGA search operations.

- Specifically, realizing $\forall ch_u \subseteq P_g$ in terms of time slots can be completed without contributing to the increase of the complexity of performing mGA on MPDP. As mentioned in Lemma 3, the complexity of *decideSequence*() is bounded by $O(MN\log N)$. Those of *estimateProbExeTime*() and *assessSchedQuality*() are both bounded by $O(N)$ as well. Namely, the solution realization process can be done by the logarithmic time in N, even though the evaluation is the most complicated phase, in order to solve MPDP by using mGA.

Therefore, we see that the proposed solution realization process dedicated to solving MPDP does not contribute to the increase of the overall complexity of mGA.

On the other hand, Lemma 2 shows that convergence to the optimum is achieved in a manner of stochastic nature. However, we note that this result does not directly indicate the probabilistic possibility of convergence, which is considered as the quality of approximation in MPDP. To this end, we focus on the fact that the convergence rate of mGA-based MJS scheme to P^{opt} is decided by recombination operators in MPDP, which adjusts the tradeoff between diversity and convergence of chromosomes as discussed in [12].

Theorem 1. *The proposed MJS scheme converges to* ch^{opt} *in probability after a finite g generations, if the recombination phase in mGA (denoted by* $rec(\cdot)$*) fulfills the following assumption: A1. Let* $\xi(P_g) = max\{Q_\lambda^{sch} : \lambda = 1, \ldots, k, \forall ch_\lambda \subseteq P_g\}$ *and* $Pr[\xi(rec(P_g)) = \xi(P_g)] = 1$*, where* $rec(P_g) = P_{g+1}$.

Proof. This assumption which is inferred from Remark 3 means that the chromosome with the highest fitness value would survive after the recombination phase performs. We also suppose that there exists a finite shortest path g_λ from an arbitrary $ch_\lambda \not\subseteq S^*$ to $ch^{opt} \subseteq S^*$, where $S^* \subseteq \mathcal{S}$ is the set of globally optimal chromosomes (i.e., solutions). Let $\hat{g} = max\{g_\lambda : \forall ch_\lambda \not\subseteq S^*\}$ and let Q^* be the globally maximum fitness value. If the probability of visiting Q^* after $\hat{g}$ generations is at least $\vartheta > 0$, then the probability that Q^* has not been found after g generations is at most $(1-\vartheta)^{\lfloor g/\hat{g} \rfloor}$.

Since A1 guarantees that the best offspring will be a parent of the next generation, it can be asserted that

$$Pr[Q^* - \xi(P_g) > 0] \leqslant (1-\vartheta)^{\lfloor g/\hat{g} \rfloor},$$

and $(1-\vartheta)^{\lfloor g/\hat{g} \rfloor}$ exponentially goes to 0 as $g \to \infty$. Therefore, Q^* will be achieved for the first time after a finite number of generations, according to Algorithms 1 and 2. □

In Theorem 1, the assumption above is reasonable because the quality of a found solution does not monotonously grow when g increases. This implies that the proposed mGA-based MJS scheme is a promising optimization method to solve MPDP. In fact, compared to the idealized situations discussed in [25], the quadratic time complexity for a super-exponentially scaled MPDP problem does not seem bad even if N is large and all jobs unrealistically experience at least one failure, that is $\mathcal{F} \geqslant N$. Because of this the computational complexity of the proposed mGA-based MJS scheme is relatively acceptable. Therefore, the proposed scheme can efficiently solve MPDP even though different rescheduling policies might be involved at the same time.

6. Performance evaluation

In this section, we have carried out a simulation study on the proposed mGA-based MJS scheme with different configurations that are unique combinations of four main factors: a used policy, a search algorithm, a distribution of job size, and a level of policy heterogeneity. For the purpose of consistency, the proposed approach will be referred to as a multihybrid job scheduling policy (MJSP), if it is necessary.

6.1. Simulation setup

For the trace-driven evaluation, we have implemented the proposed MJS scheme comprised of the two proposed processes for solution representation and realization on the real-coded mGA [17]. Then, we have integrated the MJS scheme with GridSim [26]. This simulation is designed to quantitatively evaluate the superiority of the MJS scheme in the policy-constrained DCS. For this, the average value of the corresponding metric is used to obtain every single point in resulting plots by performing the same simulation 200 times. We thus confine our focus to two different scenarios: DCS with identical policies and DCS with heterogeneous policies. These commonly include the following measures[3]:

- An size of job i, l_i, can be measured in computational units called million floating point operations (MFLOPs). Thus, the availability of resource x, c_x, is defined as the number of MFLOPs per unit time needed to process. The workload model measured and analyzed in [10,14] is used to generate a synthetic set of sequential batch jobs in consideration of statistical properties, such as job size,[4] job data size, job size distribution, and inter-arrival time.[5] The failure model used in this

[3] In order to organize the job scheduling model in DCS, these measures are injected into GridSim, in a form of the extended GridLet that specifies distribution models for jobs, resources, and failures, respectively.

[4] The variability of job sizes is expressed as a variation coefficient (K) of the bounded Pareto distribution, denoted by $BP((p_{min}, p_{max}, p_{exp})$, where p_{min} and p_{max} are the minimum and the maximum job sizes (MFLOPs), and p_{exp} is the exponent of the power law. If the parameters are given, K can be determined.

[5] For this measure, the Weibull distribution, denoted by $W(\alpha, \beta)$, is used, where parameters determine the scale and shape of the distribution, respectively [14].

Table 2
Parameter configuration in the simulation.

Parameters	Symbols	Values
Inter-arrival time	–	$W(4.25, 7.86)$
Job size	l_i	$BP(0.1 \times 10^6,\ 3 \times 10^6,\ 1.2)$
Job data size	v_i^k	0.3–0.8 MBytes
Resource availability	c_x	100–500 FLOPS
Level of resource failure	$\eta(\cdot)$	0.1–0.7
Average bandwidth	$\rho^*_{x,y}$	5–100 Mbps
Population size	$\mathcal{P}$	100
The maximum number of generations	g_{max}	500
Mutation probability	p_m	0.25
Cut-and-splice probability	p_c	0.18

simulation is based on failure trace archives [3]. From the traces, we derive the values of shape parameter β and scale parameter α, which are equal to 0.58 and 2.18, respectively. Also, the maximum number of consecutive failures for each job is set to 3 [5].

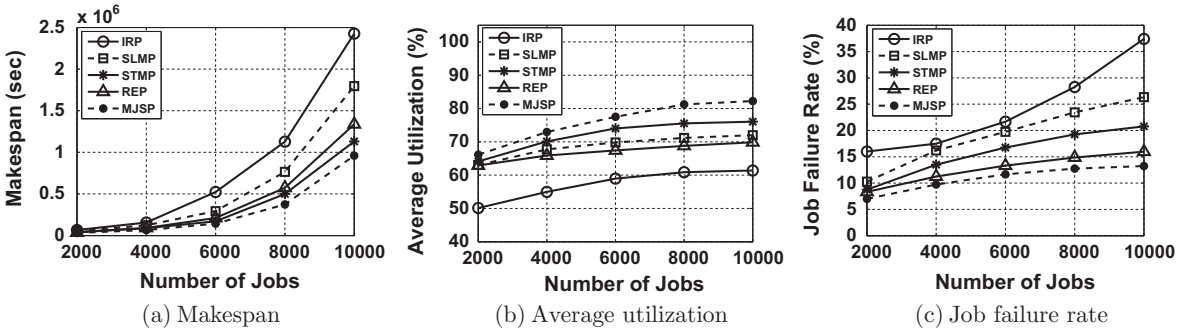
- Let $W_{i,x} = \frac{\bar{l}_{i,x}}{Max_{x=1}^{M}\{\bar{l}_{i,x}\}}$ be a weight of r_x, which refers to its computational capability relative to the fastest resource when it dedicatedly executes j_i. So, the heterogeneous capability (denoted by $\mathcal{H}$) of resources are defined as $\mathcal{H} = \frac{\sum_{x=1\sim M}(1-W_{i,x})}{M}$. We set 1 in $\mathcal{H}$, resulting in the availability of the most-capable resource being on average four times higher compared to that of the slowest resource.
- Applied mGA parameters such as mutation rate, cut-and-splice rate and population size originate from the empirical studies on genetic algorithms [12,17]. Every single point of the metrics in plots has been extensively computed by the proposed MJS scheme with 200 resources and 2000–10,000 jobs. Resources are assumed to be uniformly distributed in 10 ADs and support the rescheduling policies for fault-tolerance.

To reflect on the dynamics and complexity of MPDP, we attempt to impose randomness to the proposed MJS scheme by varying the above measures. The parameter configuration commonly used in this simulation is summarized in Table 2.

6.2. Comparison of scheduling quality

The purpose of this section is to assess the average quality of the schedules generated by mGA with the four static policies and the proposed MJSP in terms of the three metrics such as the makespan, the average utilization, and the job failure rate (as discussed in Eq. (3)) under the homogeneous policy-constrained DCS. On average, the proposed MJSP clearly improves the makespan by 115–252% and maintains the average utilization most stably, up to 82%, as shown in Fig. 4(a) and (b), respectively. Fig. 4(c) also illustrates that MJSP is the most robust approach, compared with the other policies, in coping with unexpected availability fluctuations due to failures. A proprietary allocation rule for each policy possibly causes the job interference phenomenon under precedent constraints; a wide range of network variation in terms of bandwidth may impose lots of resource fragments in availability. As a result of that, the makespan sharply increases immediately after 8000 jobs in the all policies, although the target jobs require loosely-coupled intercommunication. These facts indicate that MJSP guarantees the optimal selective use of inherent benefits of each static policy. Simulation results demonstrate the following observations.

It seems that IRP-ruled jobs suffer from severe starvation because IRP does not provide a method to fairly cope with jobs with respect to waiting time when a failure occurs. This is the main reason why IRP proves to be the worst case on all of the metrics. In SLMP and STMP, when re-dispatching terminated jobs to reserved backup resources, the migrant job should be held in a job-ready queue until the previously running job on the backup resource is finished. This feature leads to a relatively high job failure rate compared to REP and MJSP. In particular, STMP involves state-tracking overheads; however, it can reduce PET by reactivating the exact residual job only. Due to this advantage, STMP shows better performance than SLMP, although it requires additional costs for job state traces. Since REP is designed to successfully complete the jobs with high importance by executing the job replicas in parallel, it shows a less than 16% job failure rate, even with 10,000 jobs. However, its average utilization presents insufficiently low levels as compared to STMP and MJSP.



(a) Makespan (b) Average utilization (c) Job failure rate

Fig. 4. Average quality in homogeneous policy environments. IRP, SLMP, STMP, REP, and the proposed MJSP are presented for comparative study. Their results are all produced by applying our mGA. We further assume that resources support the all static policies in this simulation.

6.3. Efficiency of resource usage

With the exception of IRP, the use of the job duplication approach obviously requires more backup resources. This points to the fact that the efficiency of using redundant resources for each job can be considered as a key criterion to select an appropriate rescheduling policy in perspective of resource providers. For this reason, we will now investigate how much adding backup resources for job re-execution contributes to improvement on a particular metric for each of the individual policies. More specifically, we consider two metrics in order to evaluate a degree of efficiency in terms of makespan and average utilization, respectively. First, the resource efficiency in makespan is calculated as $\frac{makespan\ \ improvement}{mean\ \ number\ \ of\ \ resources\ \ used\ \ per\ \ job}$, where the makespan improvement is expressed by a relative ratio of makespan of used policy to that of IRP. Likewise, resource efficiency in average utilization is then also determined.

Fig. 5 reveals that resource efficiency on makespan and average utilization decreases when the expected failure probability $E[\eta]$ increases and, in general, scheduling solutions generated by the four policies tend to be much more effective in makespan (from 7.3% to 25.3%) as compared with those in average utilization (from 2.3% to 14%). When compared with REP, results from the two migration policies show better performance improvement because the parallel job execution of REP may require more resources. Furthermore, we observe that MJSP's resilience even with a high failure level ($0.5 \leqslant E[\eta] \leqslant 0.7$) is better than STMP by 4.3% and 2.9%, respectively in both metrics. In conclusion, we see that the MJSP succeeds in improving the aforementioned performance without incurring an unacceptable waste of resource usage.

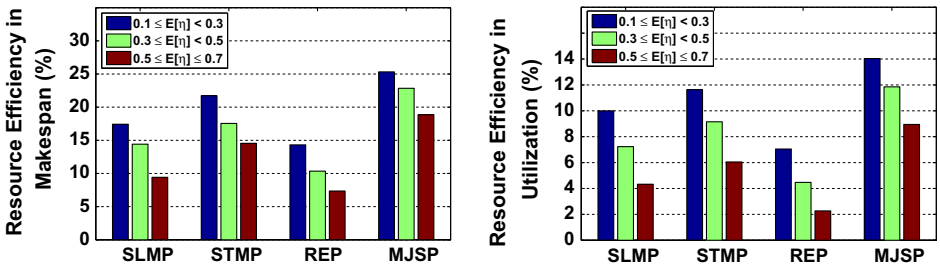
6.4. Scheduling adaptability in the policy-constrained DCS

This section aims to study how the MJS scheme guarantees the feasible performance with heterogeneous policy-based resources, adaptively supporting a limited number of policies.

Simulation setup. (*i*) We suppose that the four static policies are deployed over 10 ADs with different ratios. For example, a ratio of 1:3:1 indicates that 20%, 60%, and 20% of resources only adopt IRP, migration policies (SLMP and STMP), and REP, respectively. We thus set the three types of policy configurations *pconfig* 1 (3:1:1), *pconfig* 2 (1:3:1), and *pconfig* 3 (1:1:3) in order to take policy-constrained DCS into account. (*ii*) To investigate potential influences of different job size distributions on scheduling performance, we further classify a corresponding configuration (short jobs: long jobs) into three classes such as *jclass* 1 (8:2), *jclass* 2 (5:5), and *jclass* 3 (2:8) as discussed in [23,17]. (*iii*) To evaluate the quality of generated solutions, we then compare the proposed MJSP, based on mGA (mGA-MJSP), with a minmin algorithm [13] based MJSP, as an alternative scheme (minmin-MJSP).

Results. From Fig. 6, we observe that mGA-MJSP shows lower sensitivity to changes in job size distribution compared to minmin-MJSP in every category of policy constraints. To be precise, both schemes reach a peak value in makespan at *jclass* 3 because the total amount of workloads increase as more long jobs are offered. On the other hand, their average utilizations decrease, while the job failure rates sharply increase with more long jobs. The reason for this is that a variable degree of job size distributions makes the allocation schemes harder to achieve a fairly balanced resource occupation. When IRP is dominantly adopted by resources (*pconfig* 1), mGA-MJSP demonstrates slightly better quality than minmin-MJSP in the three metrics. Also, mGA-MJSP in its entirety seems more robust with an increase of the number of large jobs. In case of *pconfig* 2, schedules produced by mGA-MJSP achieve the shortest makespan and the highest average utilization compared to other configurations of policy ratios. For *pconfig* 3, the decreasing rate of the average utilization is smaller than that shown in *pconfig* 2, while a job failure rate is most stably maintained between 10.1% and 17.4% in this scenario.

From the above discussion, the following conclusions can be drawn. Although the performance of mGA-MJSP regarding makespan is not always significantly better than minmin-MJSP, its superiority in the maximization of resource utilization as well as the minimization of job failure rate is directly related to its high adaptability to unpredictable deviations of resource capacity. This feature makes mGA-MJSP much more attractive and effective in the policy constrained DCS. We also found that provisioning of the bicriteria optimization based on the proposed MJS scheme contributes to the adaptively guaranteed



(a) Resource efficiency in makespan (b) Resource efficiency in utilization

Fig. 5. Resource efficiency assessment. The average level of resource failure denoted by $E[\eta]$ for $\mathcal{J}$ and $\mathcal{R}$ ranges from 0.1 and 0.7 for estimating resource efficiency under different risk scales. Also, this simulation is conducted with 10,000 jobs and 200 resources.

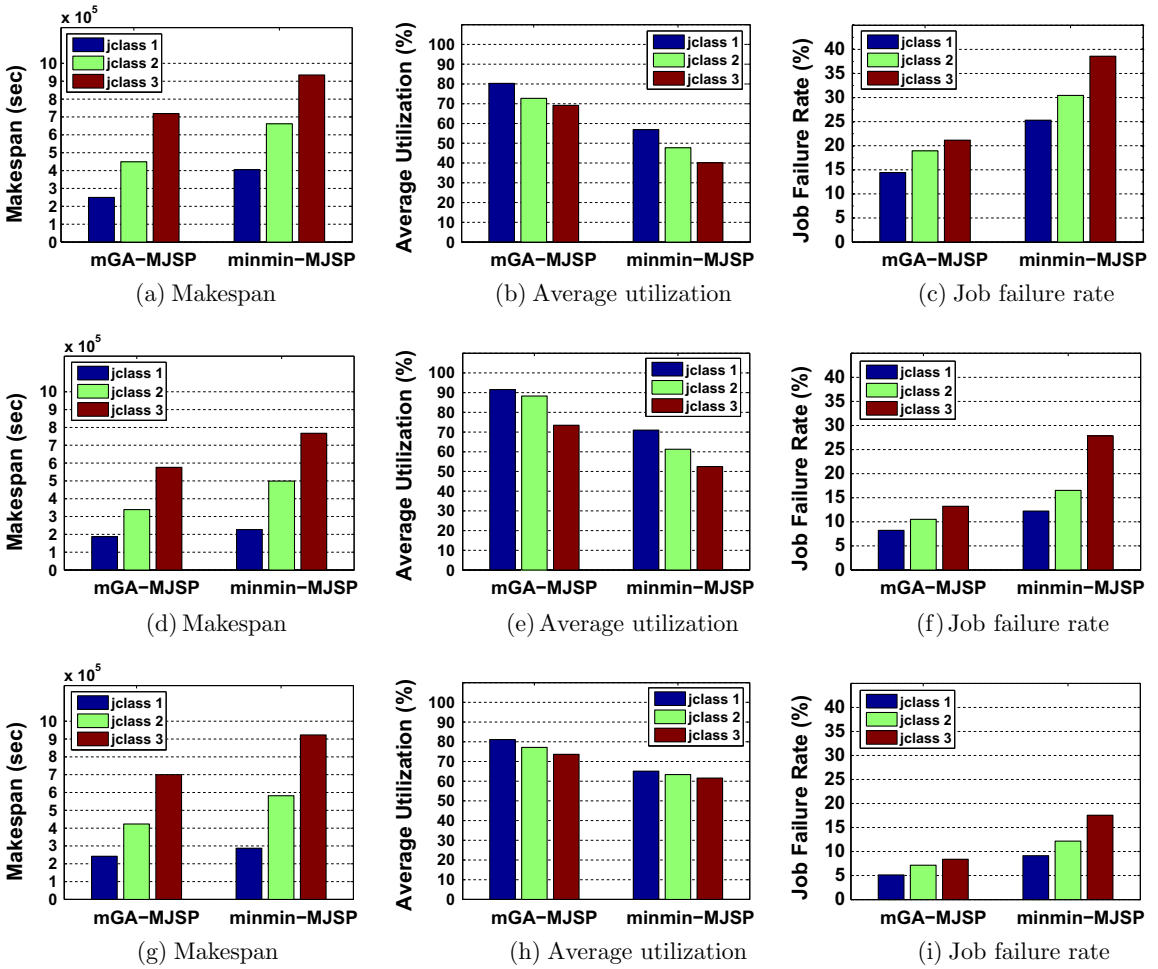
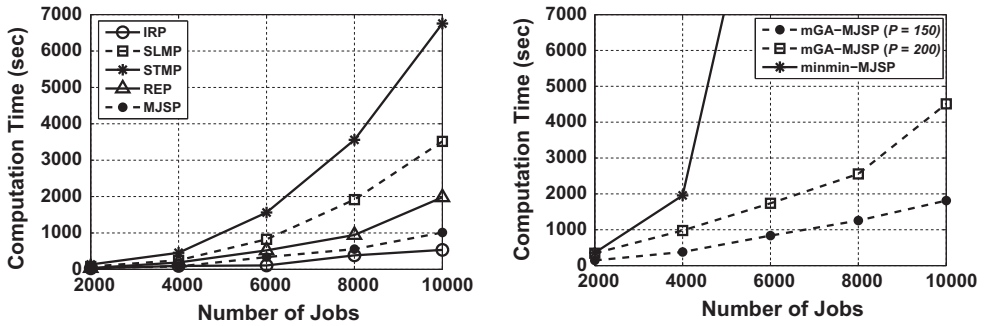


Fig. 6. Scheduling adaptability under *pconfig* 1, *pconfig* 2, and *pconfig* 3. Each of the three policy configurations is related to results (a–c); (d–f); and (g–i), respectively.

performance benefits in mGA-MJSP as compared to minmin-MJSP.

6.5. Computation time verification

The results in computation times, which are obtained by conducting a simulation with the mGA-based five policies under homogeneous policy constraints, are presented in Fig. 7(a). Fig. 7(b) also plots the elapsed times of simulations for mGA-MJSP and minmin-MJSP under heterogeneous policy constraints, whose ratio for individual resources are assumed to be randomly configured. In Fig. 7(a), IRP most quickly reaches a solution, however, the quality of the solution cannot be guaranteed as discussed. SLMP and REP can find a feasible solution at least in a reasonable time; however, these policies are more sensitive to the increase of job burstiness. Despite the strength in makespan and efficiency of resource usage, STMP spends the highest computational expense in most cases as a result of the overheads imposed by job state tracking. The competitiveness of MJSP is well maintained when the job burstiness increases, as compared with other static policies. This result means that efforts of adaptively integrating policies with resources and jobs in mGA-MJSP do not increase simulation elapsed times by very much. Namely, this implication is essentially in agreement with the properties of mGA-MJSP discussed in Section 5. As shown in Fig. 7(b), the computational scalability of minmin-MJSP seems to increase exponentially as the problem size of MPDP grows because its dominant complexity is approximately bounded by $O((NF)^2|\Pi|M) \approx O(N^2M)$, if $N \gg \mathcal{F}$ or $O(N^4M)$, if $F \geqslant N$. Although this result shows that its increasing rate of minmin-MJSP is comparable with that of mGA-MJSP for small problem sizes, mGA-MJSP requires much less computation time than minmin-MJSP as N increases steeply because it can reduce the feasible search space by depending on its stochastic search operations. This observed characteristic serves as convincing evidence of describing how much more effective mGA-MJSP is in searching optimal solutions than minmin-MJSP. This result is consistent with the theoretical analysis given by Theorem 1.



(a) Computation times of the mGA-based four static policies and MJSP

(b) Computation times of the proposed mGA-MJSP and minmin-MJSP

Fig. 7. Comparison of the computation time with different job burstiness.

7. Conclusions and future work

The problem with heterogeneous rescheduling policy currently becomes a major barrier to restrict the large scale resource pooling in a cooperative manner over ADs, which employ static and proprietary administrative strategies for mitigating harmful effects of resource failures. In this paper, we have formulated the fault-tolerant job scheduling problem with heterogeneous policies into MPDP on the primary-backup fault tolerance model. To solve MPDP effectively, we first adopted mGA's stochastic operators, then proposed a MJS scheme, which consists of two special processes for solution representation and realization tailored to MPDP. Our key contribution is proposing a mGA-based MJS scheme, which not only confidently finds an optimal scheduling solution with acceptable searching complexity, but also guarantees reliable job executions despite the availability perturbations of distributed resources. We have summarized the major findings of this study in three categories as follows:

(1) Despite the NP-completeness of MPDP, the proposed mGA-based MJS scheme makes this combinatorial optimization problem highly tractable in terms of computational complexity, which is logarithmic in the best case and quadratic in the worst case with respect to *N*. As proved in Theorem 1, this fact introduces the distinct possibility of effectively finding near-optimal solutions compared to the deterministic algorithms, e.g., minmin.

(2) The exclusive use of resources (C1) and precedence constraints (C2) help to reduce search space. Also, it turns out that penalizing solutions in proportion to the degree of the deadline violation (C3) is effective and well-directed in order to achieve more robust solutions with respect to a job failure rate. In particular, the proposed solution realization process, consisting of three functions, can estimate PETs and evaluate the quality of solutions without contributing to the increase of the overall time complexity of mGA-based MJS scheme. As a result of these features, the proposed scheme provides a solution maximizing the benefit of selective use of static rescheduling policies even for large problem sizes of MPDP.

(3) Simulation results show the superiority of the mGA-based MJS scheme compared with other static rescheduling policies such as IRP, SLMP, STMP, and REP in terms of three main metrics over DCS with identical policies. Also, we have compared the scheduling adaptability of the mGA-MJSP and minmin-MJSP in the policy-constrained DCS. The mGA-MJSP makes full use of each resource's capacity and highly improves the search efficiency compared to minmin-MJSP in every category of policy constraints.

Therefore, our approach will be especially useful when allocating the sequential batch jobs to resources adopting different types of rescheduling policies in the practical DCS with various obstacles. To the best of our knowledge, this is the first work on identifying the effects of adaptively utilizing resources governed by heterogeneous scheduling policies in the job scheduling model. Our ongoing work is directed toward implementing the mGA-based MJS scheme on a grid/cloud meta-scheduler and validating its performance feasibility in real distributed computing.

Acknowledgment

This research was supported by the ETRI R&D program of MSIP (Ministry of Science, ICT and Future Planning), Korea [12-912-06-001, Development of the Security Technology for MTM-based Mobile Devices and Next Generation Wireless LAN].

References

[1] Jorge E. Pezoa, Sagar Dhakal, Majeed M. Hayat, Maximizing service reliability in distributed computing systems with random node failures: theory and implementation, IEEE Trans. Parallel Distrib. Syst. 21 (10) (2010) 1531–1544.

[2] H. Li, D. Groep, L. Wolters, J. Templon, Job failure analysis and its implications in a large-scale production grid, in: Proc. the 2nd IEEE International Conference on e-Science and Grid Computing, December 2006, pp. 1–8.

[3] Derrick Kondo, Bahman Javadi, Alexandru Iosup, Dick Epema, The failure trace archive: enabling comparative analysis of failures in diverse distributed systems, in: Proc. the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID'10), pp. 398–407.
[4] Yulai Yuan, Yongwei Wu, Qiuping Wang, Guangwen Yang, Weimin Zheng, Job failures in high performance computing systems: a large-scale empirical study, Comput. Math. Appl. 63 (2012) 365–377.
[5] Jia-Chun Lin, Fang-Yie Leu, Ying-Ping Chen, Waqaas Munawar, Impact of MapReduce task re-execution policy on job completion reliability and job completion time, in: IEEE 28th International Conference on Advanced Information Networking and Applications (AINA'14), May 2014, pp. 712–718.
[6] Y. Zhang, H. Franke, J.E. Moreira, A. Sivasubramaniam, An integrated approach to parallel scheduling using gang-scheduling, backfilling, and migration, IEEE Trans. Parallel Distrib. Syst. 14 (3) (2003) 236–247.
[7] Achim Streit, A self-tuning job scheduler family with dynamic policy switching, in: Proc. the 8th International Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science (LNCS), Springer, 2002, pp. 1–23.
[8] R. Benjamin Clay, Zhiming Shen, Xiaosong Ma, Accelerating batch analytics with residual resources from interactive clouds, in: IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'13), August 2013, pp. 414–423.
[9] Mohammad Reza Hoseiny Farahabady, Young Choon Lee, Albert Y. Zomaya, Pareto-optimal cloud bursting, IEEE Trans. Parallel Distrib. Syst. 25 (10) (2014) 2670–2682.
[10] Alexandru Iosup, Ozan Sonmez, Shanny Anoep, Dick Epema, The performance of bags-of-tasks in large-scale distributed systems, in: Proc. the 17th International Symposium on High Performance Distributed Computing (HPDC'08), Boston, Massachusetts, USA, June 23–27, 2008, pp. 97–108.
[11] Varun Gupta, Michelle Burroughs, Mor Harchol-Balter, Analysis of scheduling policies under correlated job sizes, Perform. Eval. 67 (11) (2010) 1–24 (November 2010, pp. 996–1013).
[12] D.E. Goldberg, B. Korb, K. Deb, Messy genetic algorithms: motivation, analysis, and first results, Complex Syst. 3 (1989) 493–530.
[13] Tracy D. Braun, Howard Jay Siege, Noah Beck, Comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (2001) 810–837.
[14] Hui Li, Realistic workload modeling and its performance impacts in large-scale escience grids, IEEE Trans. Parallel Distrib. Syst. 21 (4) (2010) 480–493.
[15] Tran Ngoc Minh, Thoai Nam, Dick H.J. Epema, Parallel workload modeling with realistic characteristics, IEEE Trans. Parallel Distrib. Syst. 25 (8) (2014) 2138–2148.
[16] Antonios Litke, Dimitrios Skoutas, Konstantinos Tserpes, Theodora Varvarigou, Efficient task replication and management for adaptive fault tolerance in mobile grid environments, Future Gener. Comput. Syst. 23 (2) (2007) 163–178.
[17] Yong-Hyuk Moon, Chan-Hyun Youn, Integrated approach towards aggressive state-tracking migration for maximizing performance benefit in distributed computing, Cluster Comput. 16 (3) (2013) 367–378.
[18] Windows Azure, Microsoft Cloud Computing Platform. <http://msdn.microsoft.com/en-us/library/hh680901(v=pandp.50).aspx>.
[19] J.S. Plank, H. Casanova, M. Beck, J.J. Dongarra, Deploying fault tolerance and task migration with NetSolve, Future Gener. Comput. Syst. 15 (5) (1999) 745–755.
[20] J. Basney, M. Litzkow, T. Tannenbaum, M. Livny, Checkpoint and Migration of Unix Processes in the Condor Distributed Processing System, Technical Report 1346, April 1997.
[21] W.M. Jones, Network-aware selective job checkpoint and migration to enhance co-allocation in multi-cluster systems, Concurr. Comput. Pract. Exp. 21 (2009) 1672–1691.
[22] Mohamed-Slim Bouguerra, Denis Trystram, Frédéric Wagner, Complexity analysis of checkpoint scheduling with variable cost, IEEE Trans. Comput. 62 (6) (2013) 1269–1275.
[23] Brent Rood, Michael J. Lewis, Grid resource availability prediction-based scheduling and task replication, J. Grid Comput. 7 (4) (2009) 479–500.
[24] Guerraoui Rachid, André Schiper, Software-based replication for fault tolerance, Computer 30 (4) (1997) 68–74.
[25] David E. Goldberg, Philip Segrest, Finite Markov chain analysis of genetic algorithms, in: Proc. of the 2nd International Conference on Genetic Algorithms on Genetic Algorithms and their Application, L. Erlbaum Associates Inc., 1987, pp. 1–8.
[26] A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing. <http://www.buyya.com/gridsim/>.

Yong-Hyuk Moon received the B.S. degree in Computer Engineering from Dankook University, Seoul, Korea in 2003. He also received the M.S. and Ph.D. degrees in Information and Communication Engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea in 2006 and 2013, respectively. Since 2006, he is with the Software Research Laboratory in Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea. His research interests are in areas of resource allocation algorithms for fault-tolerance in distributed computing networks, such as peer-to-peer, cloud and grid.

Chan-Hyun Youn is a Professor with the Department of Electrical Engineering and is a Director of the Grid Middleware Center, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He is also a Vice President of Grid Forum Korea (OGF-KR). He received his B.S. and M.S. degrees in Electronics Engineering from Kyungpook National University, Daegu, Korea, in 1981 and 1985, respectively. He also received a Ph.D. in Electrical and Communications Engineering from Tohoku University, Japan, in 1994. Before joining the University, he worked for Korea Telecommunications (KT) as a Leader of High-Speed Networking Team. He also was a Visiting Scholar at MIT, Cambridge, USA in 2004.